Vrhovi valov

Naloga predstavlja nadaljevanje prve naloge s prejšnjega izpita (glej rešitve).

Epidemije prihajajo v valovih. V neki državi se je število okuženih po dnevih spreminjalo tako:

`okuzbe = [1, 6, 5, 0, 0, 0, 2, 8, 5, 3, 0, 5, 8, 0, 0, 0, 5, 1, 1]`

Tule vidimo štiri valove. Zanimajo nas vrhunci valov. V prvem valu je to drugi dan od treh (2, 3), v drugem prvi od štirih (1, 4), v tretjem drugi od dveh (2, 2), v četrtem prvi od treh (1, 3). Od obeh števil odštejemo 1 in ju delimo: tako izvemo, da so vrhovi ob `[1 / 2, 1 / 3, 1 / 1, 0 / 3]`. Tako odštevanje lepše opiše, kje se nahaja vrh: to se najlepše vidi v prvem valu, ko je vrh na polovici.

Če je val dolg 1, je vrh na položaju 0.

Na začetku in koncu seznama je lahko še poljubno število ničel. Med dvema valovoma je vsaj ena ničla.

Napiši funkcijo `vrhovi(okuzbe)`, ki za podani seznam dnevnih okužb vrne položaje vrhov valov. Klic `vrhovi([1, 6, 5, 0, 0, 0, 2, 8, 5, 3, 0, 5, 8, 0, 0, 0, 5, 1, 1])` torej vrne `[0.5, 0.333, 1, 0]`.

Rešitev

Tu so bili v rahli prednosti študenti, ki so si že prej ogledali rešitve prejšnjega izpita. Logika funkcije je namreč enake, tako da jo lahko kar hitro predelamo - a le, če jo razumemo. Če ne, pa ne. :)

```
def vrhovi(s):
    v_valovih = []
    dan = -1
    naj = 0
    # Spredaj in zadaj dodamo 0, da se ni potrebno ukvarjati z začetkom in koncem
    for x in [0] + s + [0]:
        if x > 0:  # Val se nadaljuje
            # Pogledamo, ali je število višje od najvišjega in v tem primeru
            # zabeležimo število okuženih in dan
            if x > naj:
                naj = x
                najdan = dan
            dan += 1
        else:  # Vala je konec
            if naj > 0:  # Če ni bil prazen...
                if dan > 1:
                    v_valovih.append(najdan / (dan - 1))
                else:  # Poseben primer, ko se je potrebno izogniti deljenju z 0
                    v_valovih.append(0)
```

```
            naj = 0
            najdan = 0
            dan = 0
    return v_valovih
```

```
vrhovi(okuzbe)
```

```
[0.5, 0.3333333333333333, 1.0, 0.0]
```

Nekaterih - mogoče celo vsi, ki so se naloge lotili in jo uspešno rešili - so raje nabrali cel val v seznam ter nato poiskali indeks maksimalnega elementa ter ga delili z dolžino seznama. Seveda gre tudi tako.

Najpodobnejši

Seve virusov primerjajo glede na markerje. Recimo, da bomo uporabljali naslednje markerje (vendar naj vaša rešitev deluje za poljuben nabor markerjev, ne le te, ki so v testih!)

```
markerji = {"ATTA", "GGT", "TTG", "TCCCTC"}
```

V genskem materialu vzorca virusa je vsak od teh markerjev prisoten ali pa ne. (Ali je prisoten enkrat ali večkrat, ne igra vloge). Podobnost sevov opazujemo glede na to, v koliko markerjih se ujemata.

Vzemimo, recimo

```
markerji =                          {"ATTA", "GGT", "TTG", "TCCCTC"}
vzorec = "GCGCATTAGCGGTCCCTCAAAGGT"  #  1      1       0       1
sev4 = "ATTAATTAATTAATTA"            #  1      0       0       0    => 2
```

Njuno ujemanje je 2: oba *vsebujeta* ATTA in oba *ne vsebujeta* TTG. (Glede preostalih dveh markerjev pa se razlikujeta.)

Napiši funkcijo `najpodobnejsi(vzorec, sevi, markerji)`, ki prejme nek vzorec, množico znanih sevov in množico markerjev. Vrniti mora tistega izmed znanih sevov, ki je najbolj podoben vzorcu. Če je enako podobnih več, naj vrne poljubnega izmed njih.

V testih je naštetih več sevov, v komentarjih pa so na enak način kot v gornjem primeru zapisane podobnosti.

Rešitev

Tu je bilo zelo zanimivo - morda bi moral napisati tudi *nekoliko žalostno* opazovati izdelke študentov. Pravilna rešitev gre prek vseh sevov in poišče tistega, ki se po markerjih najbolj ujema z vzorcem, torej

```
...
for sev in sevi:
```

```
    for marker in markerji:
        ...
```

Namesto tega so mnogi, morda celo večina, pisali

```
...
for marker in markerji:
    for sev in sevi:
        ...
```

Tako se stvar zelo zaplete; tisti, ki so uspeli priti po tej poti do cilja, so se iz jame, ki so si jo skopali, izvlekli s slovarji. Če zanki razporedimo pravilno, pa je rešitev preprosta.

```
def najpodobnejsi(vzorec, sevi, markerji):
    naj_podobnost = -1
    naj_sev = 0
    for sev in sevi:
        podobnost = 0
        for marker in markerji:
            if (marker in vzorec) == (marker in sev):
                podobnost += 1
        if podobnost > naj_podobnost:
            naj_podobnost = podobnost
            naj_sev = sev
    return naj_sev
```

Pogoj, ki preverja, ali oba vsebujeta ali ne vsebujeta markerja, je eleganten: to ali drži `marker in vzorec` mora biti enako temu, ali drži `marker in sev`. Oboje `True` ali oboje `False`. Brez tega bi pisali (in večina študentov tudi je pisala) `(marker in vzorec and marker in sev) or (marker not in vzorec and marker not in sev)`, ali pa je napisala celo dva `if`-a. Bolj žalostno, in tudi nedelujoče, pa je pisati `marker in sev and vzorec`. To deluje v angleščini, v Pythonu pa ne.

```
        podobnost = 0
        for marker in markerji:
            if (marker in vzorec) == (marker in sev):
                podobnost += 1
```

lahko nadomestimo z

```
        podobnost = sum((marker in vzorec) == (marker in sev) for marker in markerji)
```

Odtod hitro pridemo do

```
def najpodobnejsi(vzorec, sevi, markerji):
    return max(sevi,
               key=lambda sev: sum((marker in sev) == (marker in vzorec)
                                   for marker in markerji))
```

Stanje regij

V datoteki `obcine.txt` je seznam občin s številom prebivalcev in regijo, ki ji občina pripada. Podatki so ločeni z vejicami.

```
Moravče, 5354, Osrednjeslovenska
Ljubljana, 288832, Osrednjeslovenska
Koper, 51828, Primorska
Kočevje ob gozdu, 16549, Južnoslovenska
Piran, 17613, Primorska
Kamnik, 13768, Osrednjeslovenska
```

V datoteki `okuzbe.txt` je dnevno število okužb po občinah, v takšni obliki:

```
Kamnik: 80
Kočevje ob gozdu: 50
Ljubljana: 90
```

Vse občine iz druge datoteke se pojavijo tudi v prvi. Obratno ni nujno: če nekje ni novih okužb, te občine v drugi datoteki ni.

Napiši funkcijo `stanje_regij()`, ki prebere tidve datoteki in vrne slovar, katerega ključi so **vse regije, ki se pojavijo v prvi datoteki**, pripadajoča vrednost pa je delež okuženih. V gornjem primeru ključu `"Osrednjeslovenska"` pripada vrednost `(80 + 90) / (5354 + 288832 + 13768)` (število okuženih v občinah iz te regije, deljeno s število prebivalcev v vseh občinah te regije). Ključu `"Primorska"` pa `0`, ker v primorskih občinah ni bilo okužb.

(Datoteke niso priložene, ker se sestavijo kar znotraj testov.)

Rešitev

Naloga združuje dve veščini: branje datotek in uporabo slovarjev. Sicer pa je precej rutinska.

```
from collections import defaultdict

def stanje_regij():
    regije = {}
    populacija = defaultdict(int)
    for vrstica in open("obcine.txt"):
        ime, prebivalcev, regija = vrstica.strip().split(", ")
        regije[ime] = regija
        populacija[regija] += int(prebivalcev)

    okuzenost = dict.fromkeys(populacija, 0)
    for vrstica in open("okuzbe.txt"):
        obcina, okuzenih = vrstica.split(":")
        okuzenost[regije[obcina]] += int(okuzenih)
```

```
    for regija in okuzenost:
        okuzenost[regija] /= populacija[regija]

    return okuzenost
```

Komentarja je vreden klic `dict.fromkeys(populacija, 0)`. Ta sestavi slovar, katerega ključi so iz seznama/terke/ključev slovarja/niza/...česarkoli `populacija`, vse vrednosti pa so 0. Če te funkcije ne poznamo, takšen slovar pač sestavimo z

```
okuzenost = {}
for regija in populacija:
    okuzenost[regija] = 0
```

Tipična napaka, ki so jo nekateri delali pri reševanju, je, da so delili število okuženih v regiji s skupnim številom prebivalcev občin, ki so imele vsaj eno okužbo, namesto s populacijo regije. Tega nisem kaznoval preveč strogo, kaznovati pa sem vendarle moral. Če ne bi imeli testov, bi lahko rekli, da niste dovolj natančno brali navodil; glede na to, da so testi povedali, da je nekaj narobe, pa bi morali odkriti, v čem je problem.

Položaj vrha

Variacija na prvo nalogo. Tokrat imamo le seznam za en val. Zanima nas, na kateri dan je bilo največ okužb in koliko.

Napiši **rekurzivno** funkcijo `argmax(s)`, ki za podani **neprazni** seznam vrne indeks in vrednost največjega elementa. Klic `argmax([5, 4, 7, 8, 5, 1])` vrne `(3, 8)`, ker se največji element, 8, pojavi na indeksu 3. Če je največjih elementov več, vrne indeks prvega od njih.

Funkcija mora biti "prava rekurzivna funkcija": razen sebe naj ne kliče drugih funkcij, razen `len`, poleg tega naj ne vsebuje zank.

Rešitev

Če ima seznam le en element, se največji element nahaja na indeksu 0, in sicer je to kar ničti element.

Sicer poizvemo, kje se nahaja in kakšen je največji element ostanka. Če je večji od prvega, vrnemo le-tega, a z 1 večjim indeksom (ker je potrebno upoštevati še prvi element). Če je prvi element večji, pa je največji, očitno, prvi element.

```
def argmax(s):
    if len(s) == 1:
        return 0, s[0]
    naj_idx, naj = argmax(s[1:])
    if naj > s[0]:
        return naj_idx + 1, naj
    else:
```

```
        return 0, s[0]

argmax([5, 4, 7, 8, 5, 1])
```

```
(3, 8)
```

Sledilnik

Napiši razred `Sledilnik`, ki ima

- primeren konstruktor
- in metodo `nov_dan`, ki kot argument sprejme število novookuženih v tekočem dnevu; predstavljaj si, da jo pokličemo vsak dan, ko vlada sporoči sveže podatke.

Poleg tega naj ima atribute

- `naj_dnevnih` vsebuje največje doslej zabeleženo število dnevnih okužb,
- `tekoce_brez_okuzb` pove, koliko zadnjih dni že ni bilo nobene okužbe,
- `naj_brez_okuzb` pove, kako dolgo je bilo najdaljše zaporedje dni brez okužb.

Razen teh naj razred nima nobenih drugih atributov.

Poleg tega napiši razred `Sledilnik2`, ki je izpeljan iz razreda `Sledilnik` in doda še atribut

- `skupno_okuzenih`, ki shranjuje skupno število okuženih.

`Sledilnik2` naj smiselno uporabi oz. spreminja podedovane metode in doda čim manj kode.

Recimo, da izvedemo naslednji program.

```
s = Sledilnik2()
s.nov_dan(10)
s.nov_dan(15)
s.nov_dan(0)
s.nov_dan(0)
s.nov_dan(0)
s.nov_dan(0)
s.nov_dan(2)
s.nov_dan(0)
s.nov_dan(0)
```

Po tem je `s.naj_dnevnih` enak 15, `s.tekoce_brez_okuzb` je 2, `s.naj_brez_okuzb` je 4. Ker gre za `Sledilnik2`, je `s.skupno_okuzb` enak 27.

Če bi program nadaljevali z

```
s.nov_dan(3)
```

je `s.naj_dnevnih` še vedno 15, `s.tekoce_brez_okuzb` je 0, `s.naj_brez_okuzb` je še vedno 4 in `s.skupno_okuzb` enak 30.

Rešitev

```
class Sledilnik:
    def __init__(self):
        self.naj_dnevnih = 0
        self.naj_brez_okuzb = 0
        self.tekoce_brez_okuzb = 0

    def nov_dan(self, okuzenih):
        if okuzenih > 0:
            self.tekoce_brez_okuzb = 0
            if okuzenih > self.naj_dnevnih:
                self.naj_dnevnih = okuzenih
        else:
            self.tekoce_brez_okuzb += 1
            if self.tekoce_brez_okuzb > self.naj_brez_okuzb:
                self.naj_brez_okuzb = self.tekoce_brez_okuzb
```

V konstruktorju nastavimo tri atribute na ustrezne začetne vrednosti.

V `nov_dan` nas zanima, ali kdo okužen. Če je tako, je tekoče število dni brez okužb enako 0, poleg tega pa preverimo, ali gre morda celo za dnevni rekord.

Če okuženih ni, povečamo tekoče število dni brez okužb, potem pa preverimo, ali je to slučajno rekord števila dni brez okužb.

Drugi del naloge je preverjal, ali znate iz tega razreda izpeljati preprost razred in pravilno klicati podedovane metode.

```
class Sledilnik2(Sledilnik):
    def __init__(self):
        super().__init__()
        self.skupno_okuzenih = 0

    def nov_dan(self, okuzenih):
        super().nov_dan(okuzenih)
        self.skupno_okuzenih += okuzenih
```